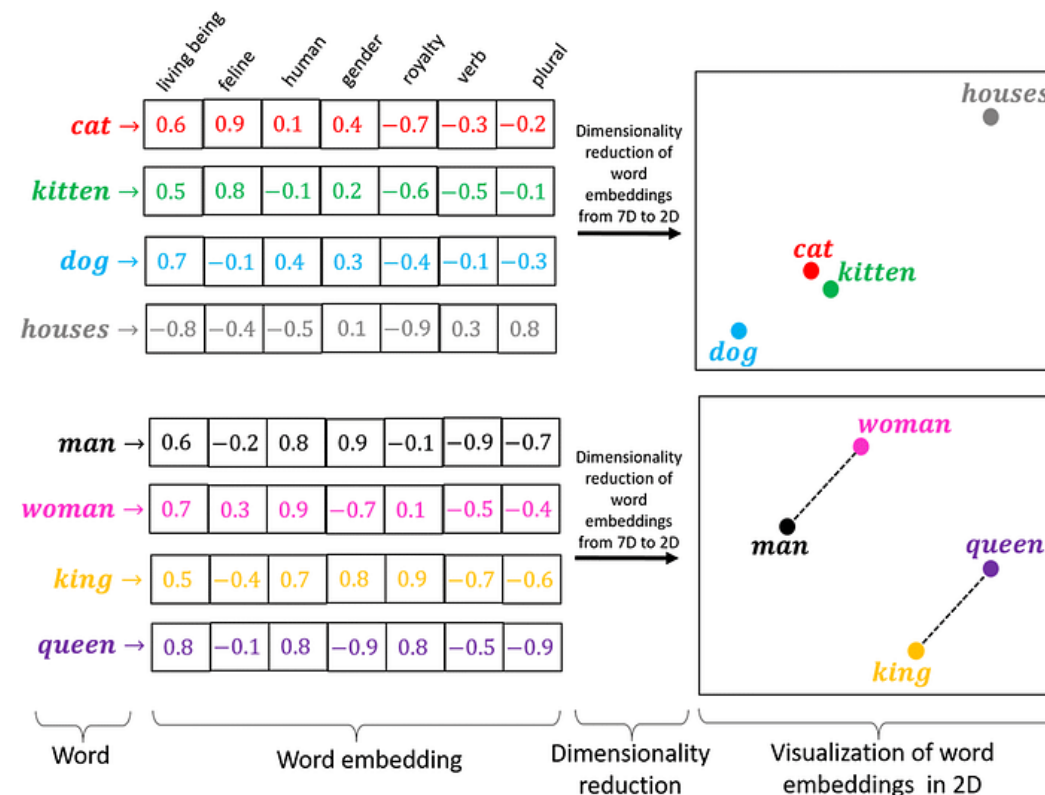


Introduction to Word Embeddings

307307 BI Methods

Word Embeddings

- Word embeddings are numerical representations of words as vectors in a multi-dimensional space, used in Natural Language Processing (NLP) to capture semantic and syntactic relationships.
- By assigning a unique vector of continuous values to each word, these embeddings allow computers to process language by understanding that words with similar meanings have similar vectors.
- This enables machines to perform complex tasks like translation, sentiment analysis, and question answering more effectively than with simple representations like one-hot encoding.



How Did We Represent Words (Before-2013)

- Traditional models like Bag-of-Words (BoW) or TF-IDF, treat words as independent, ignoring semantic similarity.
- **One-hot encoding:** Sparse, binary vectors (dimension = vocabulary size)
- Example: "king" and "queen" are as unrelated as "king" and "banana" in BoW.

Word	Dimension 1 (cat)	Dimension 2 (dog)	Dimension 3 (fish)	Dimension 4 (bird)
cat	1	0	0	0
dog	0	1	0	0
fish	0	0	1	0
bird	0	0	0	1

The Evolution of Word Representations

- **Problem:** How do we represent meaning mathematically?
- **Solution:** Distributional hypothesis - "You shall know a word by the company it keeps" (J.R. Firth, 1957)
- J.R. Firth **did not** provide a detailed technical implementation like an algorithm or computational method. His statement was more of a **linguistic philosophy** or a **theoretical principle**, not a specific engineering method.
- And much later, it inspired the **distributional hypothesis** in computational linguistics, especially by scholars like Zellig Harris and later computational models (Word2Vec, etc.)



...government debt problems turning into **banking** crises as happened in 2009...
...saying that Europe needs unified **banking** regulation to replace the hodgepodge...
...India has just given its **banking** system a shot in the arm...

These **context words** will represent **banking**

Word2Vec (Tomas Mikolov et al., 2013)

- Developed by Tomas Mikolov and team at Google.

Key Innovation

- Transformed NLP by creating dense vector representations through prediction-based models

Two Architectures

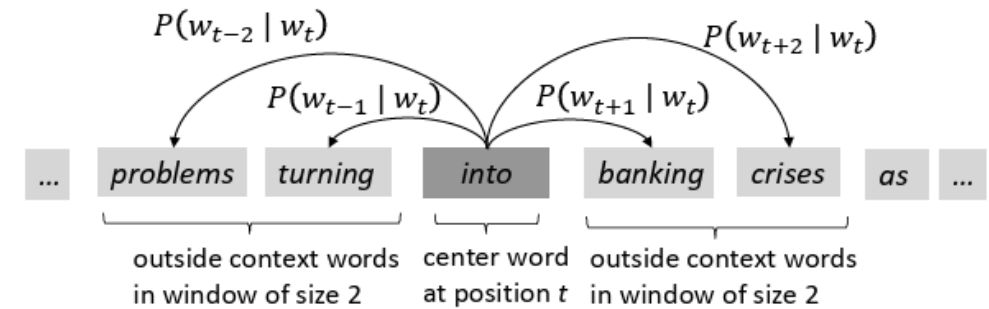
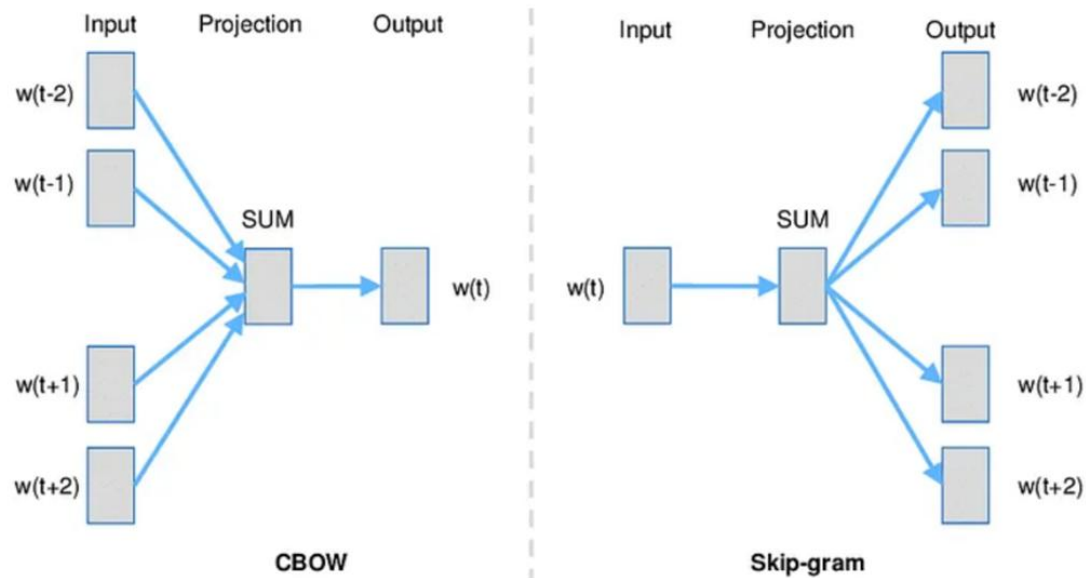
- **Continuous Bag of Words (CBOW):**
 - Predicts target word from context words
 - Faster training, better for frequent words
- **Skip-gram:**
 - Predicts context words from target word
 - Better for rare words, captures more semantic information

Characteristics

- Uses **shallow neural networks** and trains on local context windows.
- Typically, 100-300 dimensions (vs. vocabulary size)
- Linear relationships: king - man + woman \approx queen
- Efficient training through negative sampling
- **Limitations: Fixed vectors, one vector per word regardless of context**



CBow and Skip-Gram Models

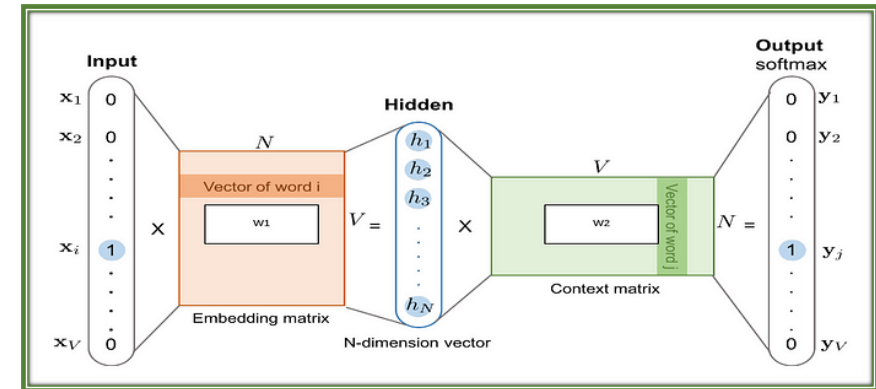


For each position $t=1, \dots, T$, predict context words within a window of fixed size m , given center word w_t .

Skip-gram Architecture (Word2Vec)

This diagram illustrates how Word2Vec's **Skip-gram model** works:

- **Input:** A one-hot encoded vector for the center word (word i).
- **Embedding Matrix:** Multiplies the input vector to produce a **dense embedding** (N -dimensional vector) — this becomes the **vector representation of the input word**.
- **Context Matrix:** The dense vector is then multiplied with another matrix to predict surrounding context words via softmax output.
- **Output:** A probability distribution over the vocabulary, aiming to maximize the likelihood of actual context words.
- This training process helps learn **meaningful word vectors** based on how words appear in context.



<https://python.plainenglish.io/understanding-word-embeddings-tf-idf-word2vec-glove-fasttext-996a59c1a8d3>

The Result of the Training Process - Similar Words -> Similar Vectors

- Visual example of how embedding changes before and after training. `sports` and `exercise` have similar embedding value post training because they are closely related

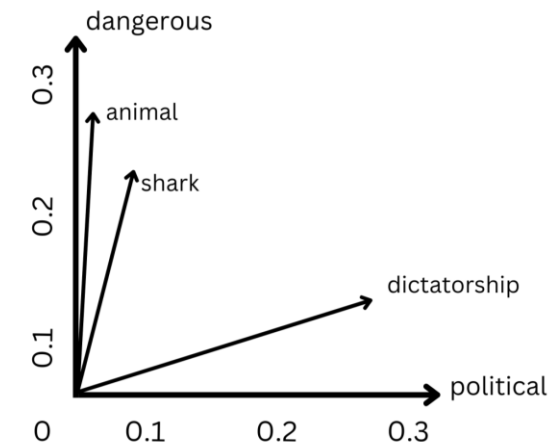


Vector Spaces and Word Embeddings

What is a Vector Space?

- A mathematical space where each word is represented as a point (or vector) in multi-dimensional space.
- Words are encoded as dense numerical vectors instead of one-hot or sparse representations (**Word Embeddings**) e.g., "king" \rightarrow [0.21, 0.72, ..., ..., 0.35]
- Word Embeddings captures **semantic** and **syntactic** relationships about/between words.
- Each dimension potentially captures semantic meaning.
- These vectors are learned from text by models like Word2Vec or GloVe.

Word	Dimension 1 (political)	Dimension 2 (dangerous)
shark	0.05	0.22
animal	0.03	0.25
dangerous	0.07	0.32
political	0.31	0.04
dictatorship	0.28	0.15



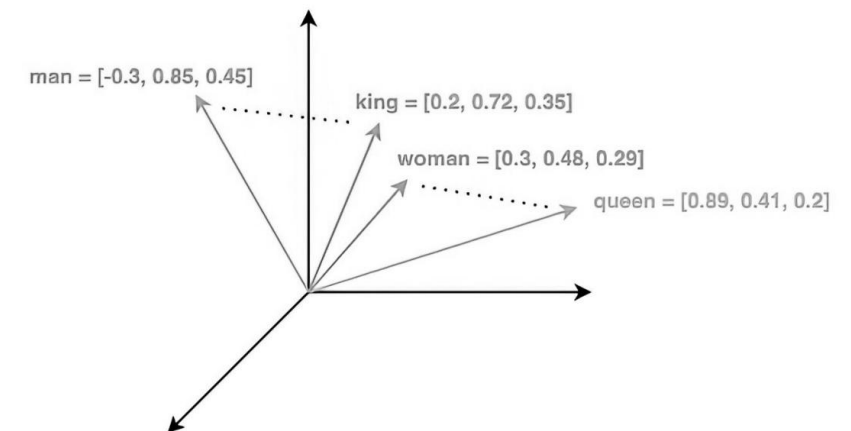
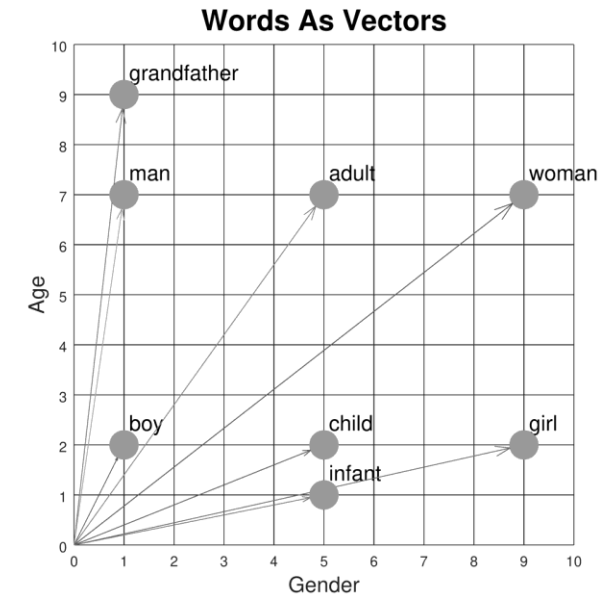
Vector Spaces and Word Embeddings

Why Use a Vector Space?

- Makes it possible to compare, visualize, and manipulate meanings of words using math.
- Enables operations like:
 - Similarity: "king" is close to "queen"
 - Analogy: "king" - "man" + "woman" \approx "queen"

Properties of Vector Space

- Semantic relationships are preserved (e.g., "shark" is closer to "dangerous" than "political").
- Similar meanings \rightarrow closer vectors.
- Dissimilar meanings \rightarrow vectors farther apart.



Glove (Pennington, Socher, Manning 2014)

- Developed by Stanford NLP Group (Pennington, Socher, Manning)

- Key Innovation**

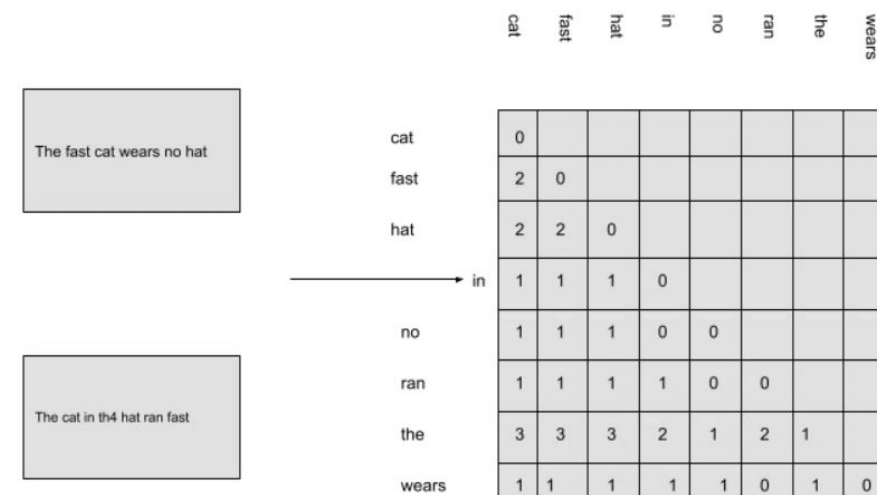
Bridges the gap between count-based methods and prediction-based methods by using global co-occurrence statistics to learn word vectors

- Approach**

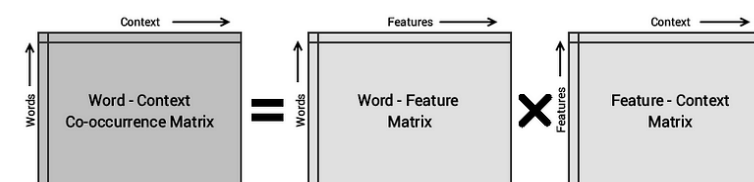
- Builds a word-word co-occurrence matrix over a large corpus
- Learns embeddings by factorizing the matrix using a weighted least squares objective

Characteristics

- Captures global statistical information while maintaining useful properties of local context
- Produces dense word vectors (typically 100–300 dimensions)
- Linear relationships in vector space are preserved: king - man + woman \approx queen
- Trained on massive corpora (Wikipedia, Common Crawl)
- Limitations: Ignores context variability—still one vector per word regardless of usage**



	cat	fast	hat	in	no	ran	the	wears
cat	0							
fast	2	0						
hat	2	2	0					
in	1	1	1	0				
no	1	1	1	0	0			
ran	1	1	1	1	0	0		
the	3	3	3	2	1	2	1	
wears	1	1	1	1	1	0	1	0



Measuring Similarity Between Word Vectors

Why Compare Word Vectors?

- Word embeddings map words into a vector space.
- **Words with similar meanings** are placed **close together** in that space.
- To quantify this "closeness," we use **vector similarity**.

Cosine Similarity

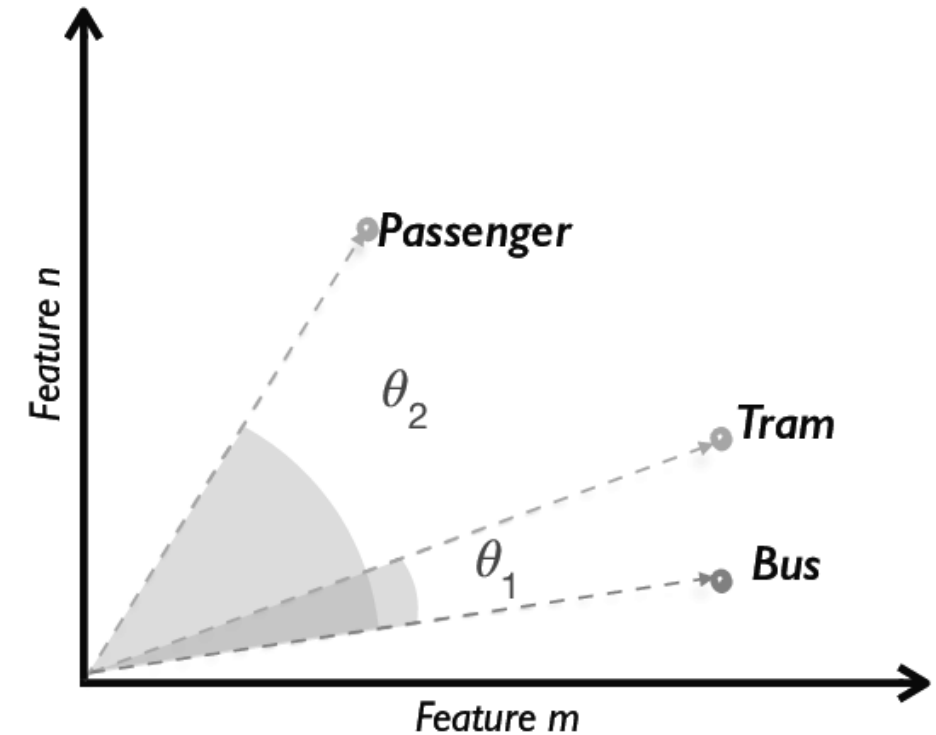
Most common metric used to compare word vectors:

$$\text{cosine_similarity}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$$

- Measures the **angle** between two vectors (not their magnitude).
- Ranges from **-1 to 1**:
 - 1 → Same direction (very similar)
 - 0 → Orthogonal (unrelated)
 - -1 → Opposite directions (very different)

Intuition

- Vectors for "king" and "queen" will have high cosine similarity.
- Vectors for "apple" and "keyboard" will have low similarity.



Experimenting with Fake Embeddings

```
import numpy as np

from sklearn.metrics.pairwise import cosine_similarity

# Fake word vectors (3D for simplicity)
word_vectors = {
    "king": np.array([0.8, 0.65, 0.1]),
    "queen": np.array([0.78, 0.66, 0.12]),
    "man": np.array([0.9, 0.1, 0.1]),
    "woman": np.array([0.88, 0.12, 0.12]),
    "apple": np.array([0.1, 0.8, 0.9]),
}

def similarity(w1, w2):
    return cosine_similarity([word_vectors[w1]], [word_vectors[w2]])[0][0]

print("Similarity(king, queen):", similarity("king", "queen"))
print("Similarity(man, woman):", similarity("man", "woman"))
print("Similarity(king, apple):", similarity("king", "apple"))
```

Use Pre-Trained Embeddings

Gensim

- Gensim is a powerful open-source Python library designed specifically for unsupervised topic modeling and natural language processing tasks, with a strong focus on working with large corpora.
- It excels in handling word embeddings and semantic similarity, offering efficient implementations of models like Word2Vec, FastText, and Doc2Vec.
- Gensim is known for its memory-efficient, streaming-based approach, which allows it to process text data without loading everything into memory.
- This makes it especially useful for working with real-world, large-scale text data.

```
import gensim.downloader as api
from gensim.models import Word2Vec

# Load pre-trained Word2Vec model
word2vec_model = api.load("word2vec-google-news-300")

# Find similar words
similar_words = word2vec_model.most_similar('computer', topn=5)
print("Words similar to 'computer':", similar_words)

# Word analogies
result = word2vec_model.most_similar(positive=['woman', 'king'],
                                     negative=['man'], topn=1)
print("king - man + woman =", result)

# Train your own Word2Vec model
sentences = [["cat", "say", "meow"], ["dog", "say", "woof"]]
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1,
                 workers=4)

# Get vector for a word
cat_vector = model.wv['cat']
print("Vector for 'cat':", cat_vector[:5]) # Show first 5 dimensions
```

Applications of Word Embeddings in NLP

1. Semantic Similarity

Measure how similar two words, phrases, or documents are by comparing their vector representations.
Example: Identifying that "doctor" and "physician" are closely related.

2. Text Classification

Used as input features for tasks like spam detection, sentiment analysis, and topic classification.
Embeddings provide rich, dense input for machine learning models.

3. Named Entity Recognition (NER)

Help identify proper nouns and classify them into categories like person, location, or organization.
Embedding-based models improve contextual understanding of named entities.

4. Machine Translation

Map words from one language to another by aligning embeddings in multilingual space.
Improves translation accuracy by leveraging semantic proximity.

5. Question Answering & Chatbots

Used to understand queries and match them with appropriate answers or responses.
Enable bots to interpret intent and context more accurately.

- **6. Information Retrieval**

Enhance search engines by retrieving results based on semantic meaning, not just keyword matches.
Example: Searching for "heart attack" returns documents containing "cardiac arrest."